



Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ**

НА ТЕМУ:

**АНАЛИЗ ЭФФЕКТИВНОСТИ ОБРАБОТКИ
ЗАПРОСОВ ПРИ РАЗНЫХ ИНДЕКСАХ ДЛЯ БД**

Студент ИУ5И-35М
(Группа)

(Подпись, дата)

М.Ю. Попов
(И.О.Фамилия)

Руководитель

(Подпись, дата)

Ю.Е. Гапанюк
(И.О.Фамилия)

Консультант

(Подпись, дата)

(И.О.Фамилия)

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой _____
(Индекс)
_____ В.Е. Терехов
(И.О.Фамилия)
« _____ » _____ 20 _____ г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме АНАЛИЗ ЭФФЕКТИВНОСТИ ОБРАБОТКИ ЗАПРОСОВ ПРИ РАЗНЫХ ИНДЕКСАХ
ДЛЯ БД

Студент группы ИУ5И-35М

Попов Михаил Юрьевич
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

учебная

Источник тематики (кафедра, предприятие, НИР) Кафедра ИУ5

График выполнения НИР: 25% к 4 нед., 50% к 8 нед., 75% к 12 нед., 100% к 16 нед.

Техническое задание Данное исследование посвящено анализу эффективности различных
индексов, таких как В-дерево, Хеш, GiST, SP-GiST, GIN и BRIN, в контексте базы данных.

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 26 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания « _____ » _____ 2024 г.

Руководитель НИР

(Подпись, дата) Ю.Е. Гапанюк
(И.О.Фамилия)

Студент

(Подпись, дата) М.Ю. Попов
(И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Оглавление

Введение	4
1. Анализ возможностей поиска в реляционной БД PostgreSQL	5
2. Создание Базы данных PostgreSQL.....	6
3. Выполнение различных запросов поиска (без индекса и с разными индексами).....	9
4. Экспериментальная оценка	18
5. Анализ эффективности	21
Заключение	22
Список литературы	23

Введение

Сегодня все больше сервисов используют запросы к большим базам данных. Примерами таких сервисов являются поисковые системы (например, Яндекс, Google и Baidu), онлайн-сервисы баз данных (например, HUAWEI Cloud и Oracle Database) и сервисы для обработки бизнес-запросов (например, онлайн-регистрация и авторизация). Поэтому эффективное управление большими базами данных является одной из главных проблем в современном мире [1-3].

Высокая скорость отклика и быстрый доступ к данным - это важные критерии при проектировании базы данных. С ростом объема данных скорость ответа на запросы становится одной из главных проблем, которые необходимо решать в системе. Индексирование - это часто наиболее удобный, простой и эффективный способ повышения производительности запросов к базе данных. Большинство проблем с производительностью запросов к базе данных можно решить с помощью индексации.

Производительность системы баз данных непосредственно зависит от скорости запроса и обработки данных. Создание правильного индекса является наиболее распространенным методом настройки базы данных для повышения скорости запроса и обработки данных. Использование индексов может значительно сократить количество операций ввода-вывода с диска, что является важным фактором для определения скорости доступа. Создание правильного и эффективного индекса может существенно повысить производительность базы данных, ускорить запрос данных и уменьшить время отклика системы.

1. Анализ возможностей поиска в реляционной БД PostgreSQL

PostgreSQL, как одна из самых продвинутых и широко используемых реляционных баз данных с открытым исходным кодом, предоставляет широкий спектр возможностей для выполнения запросов и поиска данных. Популярность PostgreSQL обусловлена ее мощными функциями, гибкостью и масштабируемостью. Возможности поиска являются ключевым компонентом данной системы, и PostgreSQL предоставляет несколько уровней поиска, включая полнотекстовый поиск, поиск по индексам и расширенные функции для обработки запросов. Полнотекстовый поиск в PostgreSQL позволяет выполнять запросы по текстовым данным, находить релевантные строки и оценивать их по значению релевантности. Это возможно благодаря использованию различных методов индексирования, таких как GIN (Generalized Inverted Index) и GiST (Generalized Search Tree), которые обеспечивают быстрый и эффективный доступ к данным. Кроме того, PostgreSQL поддерживает множество встроенных функций и операторов для работы с текстом, что позволяет легко интегрировать поиск в существующие приложения. Помимо полнотекстового поиска, PostgreSQL может похвастаться разнообразными индексами, такими как B-tree, Hash, SP-GiST и другие, которые обеспечивают оптимальную производительность для различных типов запросов. Индексация данных является важным аспектом при обработке крупных объемов информации и обеспечивает быстрое извлечение релевантных данных.

PostgreSQL предоставляет разнообразные типы индексов, среди которых самые популярные — это B-tree индексы, которые подходят для большинства операций поиска по ключам и сравнениям. Создание индекса в PostgreSQL выполняется с помощью команды `CREATE INDEX`, и такой индекс может быть одноколоночным или многоколоночным. Помимо традиционных B-tree индексов, PostgreSQL поддерживает такие структуры,

как Hash индексы, которые эффективны для равенства сравнения, но не поддерживают диапазоны, что делает их менее гибкими для большинства операций. GiST (Generalized Search Tree) индексы и GIN (Generalized Inverted Index) индексы предоставляют больше возможностей для индексации сложных типов данных, таких как геометрические или полнотекстовые данные. Например, GIN индексы особенно полезны для полнотекстового поиска и JSON полей, где требуется высокая производительность поиска по ключам и значениям внутри документов. PostgreSQL также поддерживает функциональные индексы, что позволяет создавать индексы на основе вычисляемых выражений, например, если часто требуется выполнение поиска по нижнему регистру строки, можно создать индекс на выражении `lower (columnName)`. Для оптимизации запросов PostgreSQL использует систему анализа и планирования запросов, которая автоматически выбирает наиболее подходящие индексы и стратегию выполнения запроса на основе статистики данных.

2. Создание Базы данных PostgreSQL

В первую очередь сформируем набор данных для заполнения БД. Часть используемых данных представлена на рисунке 1.



```
( 76001 , ' Alaa Abdelnaby ' , ' Alaa ' , ' Abdelnaby ' , 0 ),
( 76002 , ' Zaid Abdul-Aziz ' , ' Zaid ' , ' Abdul-Aziz ' , 0 ),
( 76003 , ' Kareem Abdul-Jabbar ' , ' Kareem ' , ' Abdul-Jabbar ' , 0 ),
( 51 , ' Mahmoud Abdul-Rauf ' , ' Mahmoud ' , ' Abdul-Rauf ' , 0 ),
( 1505 , ' Tariq Abdul-Wahad ' , ' Tariq ' , ' Abdul-Wahad ' , 0 ),
( 949 , ' Shareef Abdur-Rahim ' , ' Shareef ' , ' Abdur-Rahim ' , 0 ),
( 76005 , ' Tom Abernethy ' , ' Tom ' , ' Abernethy ' , 0 ),
( 76006 , ' Forest Able ' , ' Forest ' , ' Able ' , 0 ),
( 76007 , ' John Abramovic ' , ' John ' , ' Abramovic ' , 0 ),
( 203518 , ' Alex Abrines ' , ' Alex ' , ' Abrines ' , 0 ),
( 1630173 , ' Precious Achiuwa ' , ' Precious ' , ' Achiuwa ' , 1 ),
( 101165 , ' Alex Acker ' , ' Alex ' , ' Acker ' , 0 ),
( 76008 , ' Donald Ackerman ' , ' Donald ' , ' Ackerman ' , 0 ),
( 76009 , ' Mark Acres ' , ' Mark ' , ' Acres ' , 0 ),
( 76010 , ' Charles Acton ' , ' Charles ' , ' Acton ' , 0 ),
( 203112 , ' Quincy Acy ' , ' Quincy ' , ' Acy ' , 0 ),
```

Рисунок 1 – Датасет для заполнения БД

Далее создадим таблицы БД используя встроенную среду PgAdmin и язык SQL.

Описание базы данных:

Наша БД состоит из пяти таблиц -- common_player_info, game_info, play_by_play, officials, player.

Common_player_info состоит из 18 столбцов, включает 4902 записи.

Game_info состоит из 4 столбцов, включает 60796 записей.

Play_by_play состоит из 15 столбцов, включает более миллиона записей.

Officials состоит из 5 столбцов, включает 61789 записей.

Player состоит из 5 столбцов, включает 4810 записей.

Пример SQL запроса для создания таблицы Play_by_play представлен на рисунке 2.

```
1 create table play_by_play(  
2     game_id integer ,  
3     eventnum text,  
4     eventmsgtype text,  
5     period text,  
6     wctimestring text,  
7     player1_id integer,  
8     player1_name text,  
9     player1_team_id text,  
10    player1_team_city text,  
11    player1_team_nickname text,  
12    player2_id integer,  
13    player2_name text,  
14    player2_team_id text,  
15    player2_team_city text,  
16    player2_team_nickname text  
17 );
```

Рисунок 2 – SQL запрос создания таблицы Play_by_play

Далее на рисунках 3 и 4 представлены запрос для заполнения таблицы тестовыми данными и как в итоге БД выглядит в среде PgAdmin.

查询	查询历史
1	insert into play_by_play(game_id,eventnum,eventmsgtype,period ,wctimestring ,player1_id,player1_name,player1_team_id,
2	values (' 29600009 ' , ' 2 ' , ' 12 ' , ' 1 ' , ' 11:44 PM ' , ' 0 ' , ' nan ' , ' nan ' , ' nan ' , ' 0 ' , ' nan ' ,
3	(' 29600009 ' , ' 3 ' , ' 10 ' , ' 1 ' , ' 11:45 PM ' , ' 165 ' , ' Hakeem Olajuwon ' , ' 1610612745.0 ' , ' Houston ' ,
4	(' 29600009 ' , ' 4 ' , ' 5 ' , ' 1 ' , ' 11:46 PM ' , ' 722 ' , ' Corliss Williamson ' , ' 1610612758.0 ' , ' Sacram ' ,
5	(' 29600009 ' , ' 5 ' , ' 2 ' , ' 1 ' , ' 11:46 PM ' , ' 165 ' , ' Hakeem Olajuwon ' , ' 1610612745.0 ' , ' Houston ' ,
6	(' 29600009 ' , ' 7 ' , ' 4 ' , ' 1 ' , ' 11:46 PM ' , ' 17 ' , ' Clyde Drexler ' , ' 1610612745.0 ' , ' Houston ' ,
7	(' 29600009 ' , ' 6 ' , ' 2 ' , ' 1 ' , ' 11:46 PM ' , ' 17 ' , ' Clyde Drexler ' , ' 1610612745.0 ' , ' Houston ' ,
8	(' 29600009 ' , ' 8 ' , ' 4 ' , ' 1 ' , ' 11:46 PM ' , ' 165 ' , ' Hakeem Olajuwon ' , ' 1610612745.0 ' , ' Houston ' ,
9	(' 29600009 ' , ' 9 ' , ' 1 ' , ' 1 ' , ' 11:46 PM ' , ' 53 ' , ' Mario Elie ' , ' 1610612745.0 ' , ' Houston ' , ' R ' ,
10	(' 29600009 ' , ' 10 ' , ' 1 ' , ' 1 ' , ' 11:47 PM ' , ' 51 ' , ' Mahmoud Abdul-Rauf ' , ' 1610612758.0 ' , ' Sacram ' ,
11	(' 29600009 ' , ' 11 ' , ' 1 ' , ' 1 ' , ' 11:47 PM ' , ' 1074 ' , ' Matt Maloney ' , ' 1610612745.0 ' , ' Houston ' ,
12	(' 29600009 ' , ' 2 ' , ' 12 ' , ' 1 ' , ' 11:44 PM ' , ' 0 ' , ' nan ' , ' nan ' , ' nan ' , ' nan ' , ' 0 ' , ' nan ' ,
13	(' 29600009 ' , ' 3 ' , ' 10 ' , ' 1 ' , ' 11:45 PM ' , ' 165 ' , ' Hakeem Olajuwon ' , ' 1610612745.0 ' , ' Houston ' ,
14	(' 29600009 ' , ' 4 ' , ' 5 ' , ' 1 ' , ' 11:46 PM ' , ' 722 ' , ' Corliss Williamson ' , ' 1610612758.0 ' , ' Sacram ' ,
15	(' 29600009 ' , ' 5 ' , ' 2 ' , ' 1 ' , ' 11:46 PM ' , ' 165 ' , ' Hakeem Olajuwon ' , ' 1610612745.0 ' , ' Houston ' ,
16	(' 29600009 ' , ' 7 ' , ' 4 ' , ' 1 ' , ' 11:46 PM ' , ' 17 ' , ' Clyde Drexler ' , ' 1610612745.0 ' , ' Houston ' ,
17	(' 29600009 ' , ' 6 ' , ' 2 ' , ' 1 ' , ' 11:46 PM ' , ' 17 ' , ' Clyde Drexler ' , ' 1610612745.0 ' , ' Houston ' ,
18	(' 29600009 ' , ' 8 ' , ' 4 ' , ' 1 ' , ' 11:46 PM ' , ' 165 ' , ' Hakeem Olajuwon ' , ' 1610612745.0 ' , ' Houston ' ,
19	(' 29600009 ' , ' 9 ' , ' 1 ' , ' 1 ' , ' 11:46 PM ' , ' 53 ' , ' Mario Elie ' , ' 1610612745.0 ' , ' Houston ' , ' R ' ,
20	(' 29600009 ' , ' 10 ' , ' 1 ' , ' 1 ' , ' 11:47 PM ' , ' 51 ' , ' Mahmoud Abdul-Rauf ' , ' 1610612758.0 ' , ' Sacram ' ,
21	(' 29600009 ' , ' 11 ' , ' 1 ' , ' 1 ' , ' 11:47 PM ' , ' 1074 ' , ' Matt Maloney ' , ' 1610612745.0 ' , ' Houston ' ,
22	(' 29600009 ' , ' 2 ' , ' 12 ' , ' 1 ' , ' 11:44 PM ' , ' 0 ' , ' nan ' , ' nan ' , ' nan ' , ' nan ' , ' 0 ' , ' nan ' ,
23	(' 29600009 ' , ' 3 ' , ' 10 ' , ' 1 ' , ' 11:45 PM ' , ' 165 ' , ' Hakeem Olajuwon ' , ' 1610612745.0 ' , ' Houston ' ,
24	(' 29600009 ' , ' 4 ' , ' 5 ' , ' 1 ' , ' 11:46 PM ' , ' 722 ' , ' Corliss Williamson ' , ' 1610612758.0 ' , ' Sacram ' ,
25	(' 29600009 ' , ' 5 ' , ' 2 ' , ' 1 ' , ' 11:46 PM ' , ' 165 ' , ' Hakeem Olajuwon ' , ' 1610612745.0 ' , ' Houston ' ,
26	(' 29600009 ' , ' 7 ' , ' 4 ' , ' 1 ' , ' 11:46 PM ' , ' 17 ' , ' Clyde Drexler ' , ' 1610612745.0 ' , ' Houston ' ,
27	(' 29600009 ' , ' 6 ' , ' 2 ' , ' 1 ' , ' 11:46 PM ' , ' 17 ' , ' Clyde Drexler ' , ' 1610612745.0 ' , ' Houston ' ,
28	(' 29600009 ' , ' 8 ' , ' 4 ' , ' 1 ' , ' 11:46 PM ' , ' 165 ' , ' Hakeem Olajuwon ' , ' 1610612745.0 ' , ' Houston ' ,
29	(' 29600009 ' , ' 9 ' , ' 1 ' , ' 1 ' , ' 11:46 PM ' , ' 53 ' , ' Mario Elie ' , ' 1610612745.0 ' , ' Houston ' , ' R ' ,
30	(' 29600009 ' , ' 10 ' , ' 1 ' , ' 1 ' , ' 11:47 PM ' , ' 51 ' , ' Mahmoud Abdul-Rauf ' , ' 1610612758.0 ' , ' Sacram ' ,
31	(' 29600009 ' , ' 11 ' , ' 1 ' , ' 1 ' , ' 11:47 PM ' , ' 1074 ' , ' Matt Maloney ' , ' 1610612745.0 ' , ' Houston ' ,
32	(' 29600009 ' , ' 2 ' , ' 12 ' , ' 1 ' , ' 11:44 PM ' , ' 0 ' , ' nan ' , ' nan ' , ' nan ' , ' nan ' , ' 0 ' , ' nan ' ,

Рисунок 3 – SQL запрос заполнения таблицы Play_by_play

person_id	first_name	last_name	birthdate	school	country	height	weight	season_start	position	team_id	team_name	team_city	playercode	from_year	to_year	draft
1	31	Mahmoud	1969-03-09 00:00:00	Louisiana State	USA	6-1	162.0	9	Guard	1610612743	Nuggets	Denver	1610612743	1990.0	2003.0	191
2	1305	Tang	1974-11-03 00:00:00	San Jose State	France	6-6	235.0	7	Forward-Guard	1610612758	Kings	Sacramento	1610612758	1997.0	2003.0	191
3	76007	John	1919-02-09 00:00:00	Salem	USA	6-5	195.0	2	Forward	1610612031	Sonneten	Pittsburgh	HISTADO_Sonneten_Akronmaki	1946.0	1947.0	191
4	255122	Quincy	1996-10-06 00:00:00	Baylor	USA	6-7	240.0	7	Forward	0	nan	nan	255122	2012.0	2018.0	201
5	200801	Heaven	1984-04-20 00:00:00	Arizona	USA	6-6	220.0	2	Forward	0	nan	nan	200801	2004.0	2008.0	200
6	200919	Jordan	1994-07-08 00:00:00	UCLA	USA	6-9	209.0	3	Guard	1610612763	Grizzlies	Memphis	1610612763	2014.0	2015.0	201
7	912	Rafael	1964-07-22 00:00:00	Syracuse	USA	6-8	241.0	6	Forward	1610612751	Nets	New Jersey	1610612751	1986.0	1996.0	196
8	201336	Blake	1984-05-27 00:00:00	Florida State	USA	6-2	190.0	4	Guard	1610612746	Heat	Miami	1610612746	2007.0	2011.0	191
9	202274	Sebastian	1989-03-21 00:00:00	Florida State	Nigeria	7-1	252.0	3	Center	1610612761	Raptors	Toronto	1610612761	2010.0	2011.0	201
10	76019	Mark	1963-12-11 00:00:00	Duke	USA	6-8	217.0	5	Forward	1610612754	Bullets	Washington	HISTADO_Mark_Akron	1986.0	1990.0	196
11	200128	Furkan	1991-08-09 00:00:00	Galatasaray	Turkey	6-10	240.0	2	Forward-Center	1610612755	76ers	Philadelphia	1610612755	2014.0	2014.0	201
12	200746	Lamarion	1985-07-19 00:00:00	Texas Austin	USA	6-11	230.0	17	Center-Forward	1610612751	Nets	Brooklyn	1610612751	2006.0	2011.0	200
13	162946	Chris	1995-11-16 00:00:00	Kansas	USA	6-8	240.0	2	Forward	1610612757	Fox Blazers	Portland	1610612757	2016.0	2018.0	191
14	76022	Gary	1969-11-01 00:00:00	South Florida	USA	6-7	240.0	1	Forward	1610612759	Cavaliers	Cleveland	1610612759	1993.0	1993.0	191
15	1629734	Kyle	1996-10-21 00:00:00	Tennessee	Canada	6-10	216.0	2	Forward-Center	1610612748	Heat	Miami	1610612748	2019.0	2019.0	191
16	1629638	Nickel	1998-09-02 00:00:00	Virginia Tech	Canada	6-5	205.0	3	Guard	1610612750	Timberwolves	Minnesota	1610612750	2019.0	2022.0	201
17	1629660	Grayson	1995-10-08 00:00:00	Duke	USA	6-4	198.0	4	Guard	1610612749	Bucks	Milwaukee	1610612749	2018.0	2022.0	201
18	766	Jerome	1973-01-28 00:00:00	Pennsylvania	USA	6-4	184.0	2	Guard	1610612754	Pacers	Indiana	1610612754	1995.0	1996.0	196
19	2124	Mark	1976-06-27 00:00:00	Villanova	USA	6-10	235.0	11	Forward	1610612748	Heat	Miami	1610612748	2001.0	2002.0	191
20	1824	Peter	1979-04-26 00:00:00	Liberty	USA	6-10	260.0	1	Center	0	nan	nan	1824	1998.0	1998.0	191
21	732	Andreaf	1971-11-23 00:00:00	Michigan Illinois	USA	6-8	230.0	2	Forward	0	nan	nan	732	1995.0	1996.0	191
22	76034	Bob	1970-04-24 00:00:00	Michigan State	USA	6-3	200.0	1	Forward	1610612752	Knicks	New York	HISTADO_Bob_Akronmaki	1989.0	1989.0	191
23	2365	Chris	1987-07-07 00:00:00	Blinn J.C.	USA	6-10	240.0	15	Forward-Center	0	nan	nan	2365	2001.0	2016.0	191
24	101187	Alan	1982-10-16 00:00:00	Michigan State	USA	6-6	220.0	6	Forward-Guard	1610612751	Nets	Brooklyn	1610612751	2005.0	2016.0	191
25	200379	Antonio	1985-06-05 00:00:00	Memphis	USA	6-6	215.0	2	Guard	0	nan	nan	200379	2009.0	2009.0	191
26	76028	Chris	1984-04-07 00:00:00	St. Joseph (PA)	USA	6-2	200.0	3	Guard	1610612747	Lakers	Los Angeles	HISTADO_Chris_Anderson	1987.0	1970.0	191
27	76026	Daniel	1985-01-01 00:00:00	Southern California	USA	6-2	185.0	2	Guard	1610612757	Fox Blazers	Portland	HISTADO_Daniel_Anderson	1914.0	1970.0	191
28	1629147	Justin	1993-11-19 00:00:00	Virginia	USA	6-5	231.0	7	Forward-Guard	0	nan	nan	1629147	2015.0	2021.0	201
29	72	Kenny	1970-10-09 00:00:00	Georgia Tech	USA	6-1	168.0	14	Guard	0	nan	nan	72	1991.0	2004.0	191
30	200367	Kyle	1993-09-20 00:00:00	UCLA	USA	6-9	230.0	8	Forward-Guard	1610612750	Timberwolves	Minnesota	1610612750	2014.0	2022.0	201
31	96	Nick	1968-01-26 00:00:00	Illinois	USA	6-6	224.0	14	Forward-Guard	1610612753	Magic	Orlando	1610612753	1989.0	2002.0	196
32	1080	Shandon	1975-12-31 00:00:00	Georgia	USA	6-5	215.0	10	Guard-Forward	1610612752	Knicks	New York	1610612752	1996.0	2003.0	191
33	335	Willie	1967-01-08 00:00:00	Georgia	USA	6-8	200.0	8	Guard-Forward	1610612759	Spartans	San Antonio	1610612759	1988.0	1996.0	196
34	1512	Chris	1975-01-01 00:00:00	South East Missouri	Australia	7-0	240.0	2	Center	1610612742	Mavericks	Dallas	1610612742	1997.0	1999.0	191
35	21	Greg	1967-11-13 00:00:00	UNLV	USA	6-1	190.0	11	Guard	1610612752	Knicks	New York	1610612752	1991.0	2001.0	191
36	201332	John	1982-05-05 00:00:00	UNLV	Canada	6-9	245.0	11	Center	1610612748	Heat	Miami	1610612748	2007.0	2016.0	191
37	200544	Pero	1982-07-29 00:00:00	Olympiacos	Macedonia	6-11	260.0	2	Forward-Center	1610612757	Heat	Atlanta	1610612757	2013.0	2014.0	191
38	76061	Bob	1933-06-17 00:00:00	Michigan State	USA	6-8	220.0	1	Center	1610612744	Warriors	Philadelphia	HISTADO_Bob_Akronmaki	1956.0	1956.0	191
39	333	Daniel	1988-04-22 00:00:00	Fairleigh Dickinson	USA	6-1	180.0	15	Guard	1610612753	Magic	Orlando	1610612753	1994.0	2007.0	191
40	76053	Joe	1939-12-14 00:00:00	Texas A&M	USA	6-2	175.0	2	Guard	1610612759	Raptors	Ottawa	HISTADO_Joe_Akronmaki	1963.0	1963.0	191
41	2396	Carlos	1979-07-30 00:00:00	Puerto Rico	USA	6-2	200.0	10	Guard	1610612753	Magic	Orlando	1610612753	2001.0	2010.0	191
42	201389	Daniel	1988-03-25 00:00:00	Kansas	USA	6-9	235.0	10	Forward	1610612743	Nuggets	Denver	1610612743	2008.0	2017.0	200



Рисунок 4 – Итоговое отображение таблицы в среде

3. Выполнение различных запросов поиска (без индекса и с разными индексами).

А. Первый запрос

- Без индекса



```
1 explain analyze select * from play_by_play where player1_name = ' Robert Horry ';
```

Data Output 消息 通知	
	
	QUERY PLAN 
	text
1	Gather (cost=1000.00..22045.30 rows=1022 width=120) (actual time=1.157..140.569 rows=2139 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Parallel Seq Scan on play_by_play (cost=0.00..20943.10 rows=426 width=120) (actual time=0.361..48.429 rows=713 loop...
5	Filter: (player1_name = ' Robert Horry '::text)
6	Rows Removed by Filter: 282621
7	Planning Time: 0.509 ms
8	Execution Time: 140.661 ms

- В-дерево

```
1 create index on play_by_play(player1_name);
```

```
1 explain analyze select * from play_by_play where player1_name = ' Robert Horry ';
```

Data Output 消息 通知	
	
	QUERY PLAN 
	text
1	Bitmap Heap Scan on play_by_play (cost=12.35..3263.77 rows=1022 width=120) (actual time=0.257..2.280 rows=2139 loops=1)
2	Recheck Cond: (player1_name = ' Robert Horry '::text)
3	Heap Blocks: exact=690
4	-> Bitmap Index Scan on play_by_play_player1_name_idx (cost=0.00..12.09 rows=1022 width=0) (actual time=0.173..0.173 rows=2139 loop...
5	Index Cond: (player1_name = ' Robert Horry '::text)
6	Planning Time: 0.434 ms
7	Execution Time: 2.356 ms

- **Xem**

```
1 create index on play_by_play using hash(player1_name);
```

```
1 explain analyze select * from play_by_play where player1_name = ' Robert Horry ';
```

Data Output 消息 通知



	QUERY PLAN	
	text	🔒
1	Bitmap Heap Scan on play_by_play (cost=31.92..3283.34 rows=1022 width=120) (actual time=0.182..1.005 rows=2139 loops=1)	
2	Recheck Cond: (player1_name = ' Robert Horry '::text)	
3	Heap Blocks: exact=690	
4	-> Bitmap Index Scan on play_by_play_player1_name_idx (cost=0.00..31.66 rows=1022 width=0) (actual time=0.116..0.116 rows=2139 loop...)	
5	Index Cond: (player1_name = ' Robert Horry '::text)	
6	Planning Time: 0.367 ms	
7	Execution Time: 1.087 ms	

- **GIST**

```
1 create index on play_by_play using gist(player1_name);
```

```
1 explain analyze select * from play_by_play where player1_name = ' Robert Horry ';
```

Data Output 消息 通知



	QUERY PLAN	
	text	🔒
1	Bitmap Heap Scan on play_by_play (cost=44.21..3295.63 rows=1022 width=120) (actual time=0.423..1.012 rows=2139 loops=1)	
2	Recheck Cond: (player1_name = ' Robert Horry '::text)	
3	Heap Blocks: exact=690	
4	-> Bitmap Index Scan on play_by_play_player1_name_idx (cost=0.00..43.95 rows=1022 width=0) (actual time=0.368..0.369 rows=2139 loop...)	
5	Index Cond: (player1_name = ' Robert Horry '::text)	
6	Planning Time: 0.315 ms	
7	Execution Time: 1.089 ms	

- **SP-GIST**

```
1 create index on play_by_play using spgist(player1_name);
```

```
1 explain analyze select * from play_by_play where player1_name = ' Robert Horry ';
```

Data Output 消息 通知



	QUERY PLAN	
	text	🔒
1	Bitmap Heap Scan on play_by_play (cost=28.21..3279.63 rows=1022 width=120) (actual time=0.259..0.868 rows=2139 loops=1)	
2	Recheck Cond: (player1_name = ' Robert Horry '::text)	
3	Heap Blocks: exact=690	
4	-> Bitmap Index Scan on play_by_play_player1_name_idx (cost=0.00..27.95 rows=1022 width=0) (actual time=0.205..0.205 rows=2139 loop...)	
5	Index Cond: (player1_name = ' Robert Horry '::text)	
6	Planning Time: 0.370 ms	
7	Execution Time: 0.942 ms	

• GIN

查询 查询历史

```
1 create index on play_by_play using gin(player1_name);
```

Data Output 消息 通知

CREATE INDEX

耗时1 秒 578 毫秒 成功返回查询。

查询 查询历史

```
1 explain analyze select * from play_by_play where player1_name = ' Robert Horry ';
```

Data Output 消息 通知



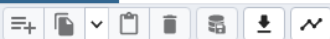
	QUERY PLAN	
	text	🔒
1	Bitmap Heap Scan on play_by_play (cost=19.92..3271.35 rows=1022 width=120) (actual time=0.303..0.938 rows=2139 loops=1)	
2	Recheck Cond: (player1_name = ' Robert Horry '::text)	
3	Heap Blocks: exact=690	
4	-> Bitmap Index Scan on play_by_play_player1_name_idx (cost=0.00..19.67 rows=1022 width=0) (actual time=0.236..0.237 rows=2139 loop...)	
5	Index Cond: (player1_name = ' Robert Horry '::text)	
6	Planning Time: 0.399 ms	
7	Execution Time: 1.036 ms	

• BRIN

```
1 create index on play_by_play using brin(player1_name);
```

```
1 explain analyze select * from play_by_play where player1_name = ' Robert Horry ';
```

Data Output 消息 通知



	QUERY PLAN	
	text	🔒
1	Gather (cost=1014.62..20152.78 rows=1022 width=120) (actual time=1.420..281.487 rows=2139 loops=1)	
2	Workers Planned: 2	
3	Workers Launched: 2	
4	-> Parallel Bitmap Heap Scan on play_by_play (cost=14.62..19050.58 rows=426 width=120) (actual time=4.811..179.607 rows=713 loops=3)	
5	Recheck Cond: (player1_name = ' Robert Horry '::text)	
6	Rows Removed by Index Recheck: 282621	
7	Heap Blocks: lossy=12529	
8	-> Bitmap Index Scan on play_by_play_player1_name_idx (cost=0.00..14.37 rows=483831 width=0) (actual time=0.423..0.424 rows=165160 loop...)	
9	Index Cond: (player1_name = ' Robert Horry '::text)	
10	Planning Time: 0.376 ms	
11	Execution Time: 281.611 ms	

В. Второй запрос

- БЕЗ ИНДЕКСА

```
1 explain analyze select * from officials where first_name = ' Steve '
```

Data Output 消息 通知



	QUERY PLAN	
	text	🔒
1	Seq Scan on officials (cost=0.00..1276.36 rows=643 width=32) (actual time=0.021..7.380 rows=656 loops...	
2	Filter: (first_name = ' Steve '::text)	
3	Rows Removed by Filter: 61133	
4	Planning Time: 0.074 ms	
5	Execution Time: 7.576 ms	

- В-дерево

```
1 create index on officials(first_name);
```

```
1 explain analyze select * from officials where first_name = ' Steve '
```

Data Output 消息 通知



	QUERY PLAN	
	text	🔒
1	Bitmap Heap Scan on officials (cost=9.27..548.21 rows=643 width=32) (actual time=0.087..0.356 rows=656 loops=1)	
2	Recheck Cond: (first_name = ' Steve '::text)	
3	Heap Blocks: exact=331	
4	-> Bitmap Index Scan on officials_first_name_idx (cost=0.00..9.11 rows=643 width=0) (actual time=0.052..0.052 rows=656 loop...	
5	Index Cond: (first_name = ' Steve '::text)	
6	Planning Time: 0.073 ms	
7	Execution Time: 0.398 ms	

- Хеш

```
1 create index on officials using hash(first_name);
```

```
1 explain analyze select * from officials where first_name = ' Steve '
2
```

Data Output 消息 通知



	QUERY PLAN	
	text	🔒
1	Bitmap Heap Scan on officials (cost=20.98..559.92 rows=643 width=32) (actual time=0.076..0.352 rows=656 loops=1)	
2	Recheck Cond: (first_name = ' Steve '::text)	
3	Heap Blocks: exact=331	
4	-> Bitmap Index Scan on officials_first_name_idx (cost=0.00..20.82 rows=643 width=0) (actual time=0.046..0.046 rows=656 loop...	
5	Index Cond: (first_name = ' Steve '::text)	
6	Planning Time: 0.079 ms	
7	Execution Time: 0.394 ms	

- GIST

```
1 create index on officials using gist(first_name);
```

```
1 explain analyze select * from officials where first_name = ' Steve '
2
```

Data Output 消息 通知



	QUERY PLAN	
	text	🔒
1	Bitmap Heap Scan on officials (cost=25.26..564.20 rows=643 width=32) (actual time=0.234..0.493 rows=656 loops=1)	
2	Recheck Cond: (first_name = ' Steve '::text)	
3	Heap Blocks: exact=331	
4	-> Bitmap Index Scan on officials_first_name_idx (cost=0.00..25.10 rows=643 width=0) (actual time=0.183..0.183 rows=656 loop...)	
5	Index Cond: (first_name = ' Steve '::text)	
6	Planning Time: 0.071 ms	
7	Execution Time: 0.537 ms	

- SP-GIST

```
1 create index on officials using spgist(first_name);
2
```

```
1 explain analyze select * from officials where first_name = ' Steve '
2
```

Data Output 消息 通知



	QUERY PLAN	
	text	🔒
1	Bitmap Heap Scan on officials (cost=17.26..556.20 rows=643 width=32) (actual time=0.100..0.350 rows=656 loops=1)	
2	Recheck Cond: (first_name = ' Steve '::text)	
3	Heap Blocks: exact=331	
4	-> Bitmap Index Scan on officials_first_name_idx (cost=0.00..17.10 rows=643 width=0) (actual time=0.070..0.071 rows=656 loop...)	
5	Index Cond: (first_name = ' Steve '::text)	
6	Planning Time: 0.317 ms	
7	Execution Time: 0.387 ms	

- GIN

```
1 create index on officials using gin(first_name);
2
```

```
1 explain analyze select * from officials where first_name = ' Steve '
2
```

Data Output 消息 通知



	QUERY PLAN	
	text	
1	Bitmap Heap Scan on officials (cost=16.98..555.91 rows=643 width=32) (actual time=0.098..0.356 rows=656 loops=1)	
2	Recheck Cond: (first_name = ' Steve '::text)	
3	Heap Blocks: exact=331	
4	-> Bitmap Index Scan on officials_first_name_idx (cost=0.00..16.82 rows=643 width=0) (actual time=0.070..0.071 rows=656 loop...	
5	Index Cond: (first_name = ' Steve '::text)	
6	Planning Time: 0.412 ms	
7	Execution Time: 0.393 ms	

- BRIN

```
1 create index on officials using brin(first_name);
2
```

```
1 explain analyze select * from officials where first_name = ' Steve '
2
```

Data Output 消息 通知



	QUERY PLAN	
	text	
1	Seq Scan on officials (cost=0.00..1276.36 rows=643 width=32) (actual time=0.017..6.127 rows=656 loops...	
2	Filter: (first_name = ' Steve '::text)	
3	Rows Removed by Filter: 61133	
4	Planning Time: 0.442 ms	
5	Execution Time: 6.158 ms	

С. Третий запрос

- Без индекса

```
1 explain analyze select * from play_by_play where player1_team_nickname = ' Kings ' ;
```

Data Output 消息 通知



	QUERY PLAN	
	text	
1	Gather (cost=1000.00..28771.34 rows=31400 width=120) (actual time=0.528..175.198 rows=31870 loops=1)	
2	Workers Planned: 2	
3	Workers Launched: 2	
4	-> Parallel Seq Scan on play_by_play (cost=0.00..24631.34 rows=13083 width=120) (actual time=0.315..89.105 rows=10623 loop...	
5	Filter: (player1_team_nickname = ' Kings '::text)	
6	Rows Removed by Filter: 322711	
7	Planning Time: 0.752 ms	
8	Execution Time: 176.384 ms	

- В-дерево

```
1 create index on play_by_play(player1_team_nickname);
```

```
1 explain analyze select * from play_by_play where player1_team_nickname = ' Kings ';
```

Data Output 消息 通知



	QUERY PLAN	
	text	
1	Bitmap Heap Scan on play_by_play (cost=351.78..20946.44 rows=31400 width=120) (actual time=0.817..6.291 rows=31870 loops=1)	
2	Recheck Cond: (player1_team_nickname = ' Kings '::text)	
3	Heap Blocks: exact=1537	
4	-> Bitmap Index Scan on play_by_play_player1_team_nickname_idx1 (cost=0.00..343.93 rows=31400 width=0) (actual time=0.646..0.647 rows=31870 loop...	
5	Index Cond: (player1_team_nickname = ' Kings '::text)	
6	Planning Time: 0.080 ms	
7	Execution Time: 7.211 ms	

- Хеш

```
1 create index on play_by_play using hash(player1_team_nickname);
```

```
1 explain analyze select * from play_by_play where player1_team_nickname = ' Kings ';
```

```
2
```

Data Output 消息 通知



	QUERY PLAN	
	text	
1	Bitmap Heap Scan on play_by_play (cost=1003.35..21598.02 rows=31400 width=120) (actual time=7.462..17.698 rows=31870 loops=1)	
2	Recheck Cond: (player1_team_nickname = ' Kings '::text)	
3	Heap Blocks: exact=1537	
4	-> Bitmap Index Scan on play_by_play_player1_team_nickname_idx (cost=0.00..995.50 rows=31400 width=0) (actual time=7.325..7.326 rows=31870 loop...	
5	Index Cond: (player1_team_nickname = ' Kings '::text)	
6	Planning Time: 0.414 ms	
7	Execution Time: 18.654 ms	


- GIST

```
1 create index on play_by_play using gist(player1_team_nickname);
```

```
1 explain analyze select * from play_by_play where player1_team_nickname= ' Kings ';
```

Data Output 消息 通知



QUERY PLAN		
	text	
1	Bitmap Heap Scan on play_by_play (cost=1215.63..21810.30 rows=31400 width=120) (actual time=4.499..11.308 rows=31870 loops=1)	
2	Recheck Cond: (player1_team_nickname = ' Kings '::text)	
3	Heap Blocks: exact=1537	
4	-> Bitmap Index Scan on play_by_play_player1_team_nickname_idx (cost=0.00..1207.79 rows=31400 width=0) (actual time=4.337..4.338 rows=31870 loop...	
5	Index Cond: (player1_team_nickname = ' Kings '::text)	
6	Planning Time: 6.492 ms	
7	Execution Time: 12.323 ms	


- SP-GIST

```
1 create index on play_by_play using spgist(player1_team_nickname);
```

```
1 explain analyze select * from play_by_play where player1_team_nickname= ' Kings ';
```

Data Output 消息 通知



QUERY PLAN		
	text	
1	Bitmap Heap Scan on play_by_play (cost=751.63..21346.30 rows=31400 width=120) (actual time=2.941..9.030 rows=31870 loops=1)	
2	Recheck Cond: (player1_team_nickname = ' Kings '::text)	
3	Heap Blocks: exact=1537	
4	-> Bitmap Index Scan on play_by_play_player1_team_nickname_idx (cost=0.00..743.78 rows=31400 width=0) (actual time=2.791..2.792 rows=31870 loop...	
5	Index Cond: (player1_team_nickname = ' Kings '::text)	
6	Planning Time: 0.362 ms	
7	Execution Time: 16.126 ms	


- GIN

```
1 create index on play_by_play using gin(player1_team_nickname);
```

```
1 explain analyze select * from play_by_play where player1_team_nickname= ' Kings ';
```

Data Output 消息 通知



QUERY PLAN		
	text	
1	Bitmap Heap Scan on play_by_play (cost=295.35..20890.02 rows=31400 width=120) (actual time=2.006..6.855 rows=31870 loops=1)	
2	Recheck Cond: (player1_team_nickname = ' Kings '::text)	
3	Heap Blocks: exact=1537	
4	-> Bitmap Index Scan on play_by_play_player1_team_nickname_idx (cost=0.00..287.50 rows=31400 width=0) (actual time=1.874..1.874 rows=31870 loop...	
5	Index Cond: (player1_team_nickname = ' Kings '::text)	
6	Planning Time: 0.651 ms	
7	Execution Time: 7.761 ms	

- **BRIN**

```
1 create index on play_by_play using brin(player1_team_nickname);
2
1 explain analyze select * from play_by_play where player1_team_nickname= ' Kings ';
2
3
```

Data Output 消息 通知

QUERY PLAN	
	text
1	Bitmap Heap Scan on play_by_play (cost=21.79..24438.27 rows=31400 width=120) (actual time=0.656..260.561 rows=31870 loops=1)
2	Recheck Cond: (player1_team_nickname = 'Kings '::text)
3	Rows Removed by Index Recheck: 968132
4	Heap Blocks: lossy=19423
5	-> Bitmap Index Scan on play_by_play_player1_team_nickname_idx (cost=0.00..13.94 rows=399478 width=0) (actual time=0.639..0.639 rows=194230 loop=1)
6	Index Cond: (player1_team_nickname = 'Kings '::text)
7	Planning Time: 0.442 ms
8	Execution Time: 261.591 ms

4. Экспериментальная оценка

У нас есть база данных, состоящая из пяти таблиц - common_player_info, game_info, play_by_play, officials, player.

Таблица Common_player_info состоит из 18 столбцов и включает 4902 записи.

Таблица Game_info состоит из 4 столбцов и включает 60796 записей.

Таблица Play_by_play состоит из 15 столбцов и включает более миллиона записей.

Таблица Officials состоит из 5 столбцов и включает 61789 записей.

Таблица Player состоит из 5 столбцов и включает 4810 записей.

Для проведения исследования мы использовали компьютер i7-7700HQ (2,80 ГГц) с процессором Intel (R) Core (TM), 16 ГБ оперативной памяти и 64-разрядной операционной системой Windows 10. Для выполнения запросов мы использовали утилиту PgAdmin 4. Язык запроса, который мы использовали, - SQL. Все результаты, представленные в таблице, являются средним значением за 20 измерений.

Мы использовали утилиту PgAdmin 4 для создания индексов и выполнения запросов в двух таблицах. Первый запрос: `SELECT * FROM play_by_play WHERE player1_name = 'Robert Horry'`. Второй запрос: `SELECT * FROM officials WHERE jersey_num > 23 AND jersey_num < 34`. В конце мы использовали оператор `EXPLAIN ANALYZE`, чтобы вывести `QUERY PLAN` запроса, и получили результаты [10].

Таблица 1. Время выполнения первого запроса, ms

	Без индекса	В-дерево	Хеш	GiST	SP- GiST	GIN	BRIN
1	173.126	4.609	3.282	5.746	3.804	3.586	196.938
2	165.101	1.072	1.334	1.469	1.251	1.127	181.639
3	164.645	1.120	1.277	1.288	1.212	1.056	161.869
4	142.084	0.977	1.198	1.561	1.139	1.124	157.087
5	145.455	1.102	1.222	1.374	1.164	1.142	181.516
6	141.479	1.136	1.214	1.388	1.218	1.133	141.103
7	161.559	1.189	1.239	1.491	1.410	1.223	173.750
8	146.068	0.966	1.799	1.403	1.173	1.316	173.758
9	142.547	0.995	1.422	1.533	1.516	1.405	272.208
10	237.617	1.368	1.716	1.821	1.513	1.065	169.944
11	222.319	0.999	1.371	1.557	1.119	1.090	277.353
12	202.847	1.193	1.349	1.427	1.096	1.599	181.426
13	206.047	1.296	1.526	1.344	1.405	1.468	152.230
14	171.365	1.079	1.404	1.522	1.577	1.104	172.429
15	180.604	1.349	1.665	1.385	1.561	1.544	179.639
16	188.487	1.097	1.410	1.950	1.112	1.169	250.849
17	129.209	1.026	1.702	1.27	1.131	1.231	156.131
18	142.897	1.152	1.152	1.417	1.184	1.117	164.660
19	144.326	0.963	1.292	1.545	1.343	1.248	159.000
20	156.613	0.968	1.296	1.249	1.523	1.081	148.673
Среднее Значение	168.219	1.282	1.494	1.687	1.422	1.341	182.610
Среднеквадрат ичное	29.131	0.773	0.450	0.946	0.571	0.539	37.920

Среднее время обработки запроса

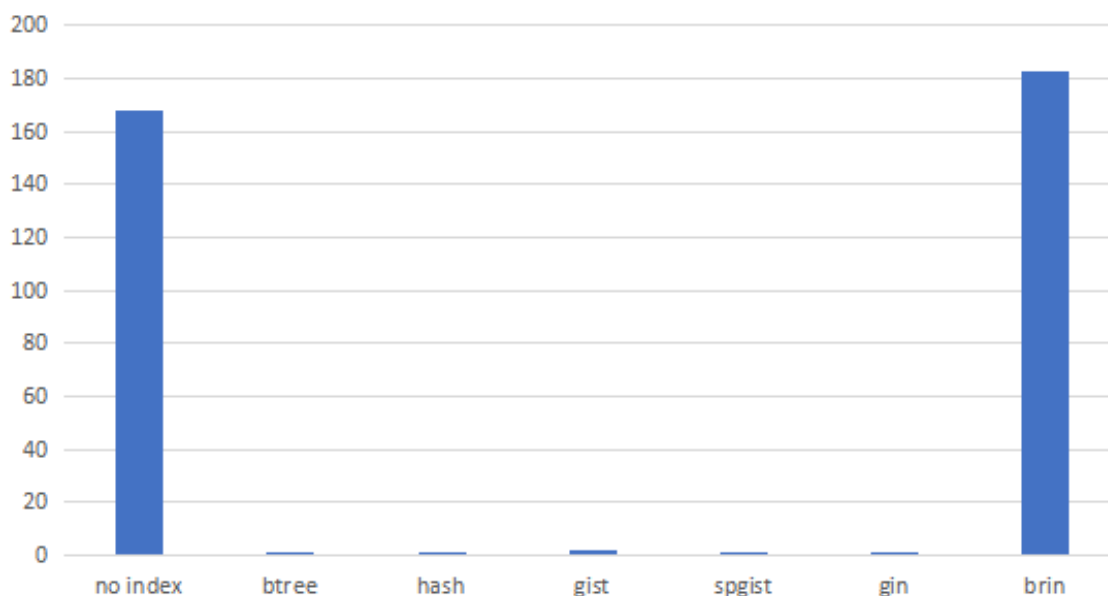


Рисунок 5 – Среднее время выполнения первого запроса

Таблица 2. Время выполнения второго запроса, ms

	Без индекса	В-дерево	Хеш	GiST	SP- GiST	GIN	BRIN
1	5.622	1.155	1.900	4.365	1.79	1.999	5.578
2	4.141	0.531	0.647	0.72	0.699	0.451	7.536
3	4.602	0.436	0.46	0.557	0.398	0.402	4.572
4	4.364	0.422	0.533	0.502	0.403	0.633	6.114
5	4.003	0.561	0.449	0.481	0.399	0.395	5.229
6	5.197	0.420	0.407	0.803	0.653	0.457	4.021
7	6.908	0.530	0.439	0.539	0.419	0.392	4.865
8	4.151	1.086	0.533	0.542	0.396	0.442	4.889
9	3.958	0.420	0.460	0.499	0.425	0.409	7.129
10	4.120	0.464	0.451	0.495	0.444	0.396	4.181
11	4.463	0.373	0.457	1.055	0.432	0.408	4.515
12	6.989	0.444	0.416	0.515	0.548	0.442	4.166
13	4.565	0.427	0.546	0.563	0.421	0.403	4.873
14	4.620	0.374	0.609	0.559	0.412	0.462	4.430
15	3.964	0.573	0.461	0.529	0.410	0.511	4.894
16	4.303	0.383	0.412	0.846	0.411	0.394	4.124
17	4.323	0.408	0.420	0.495	0.419	0.414	4.764
18	5.279	0.387	0.424	0.593	0.619	0.396	4.268
19	3.961	0.555	0.798	0.553	0.434	0.412	4.184
20	4.449	0.541	0.407	0.564	0.440	0.404	4.234
Среднее Значение	4.699	0.525	0.561	0.789	0.462	0.511	4.928
Среднеквадрат ичное	0.871	0.209	0.322	0.833	0.091	0.346	0.955

Среднее время обработки запроса

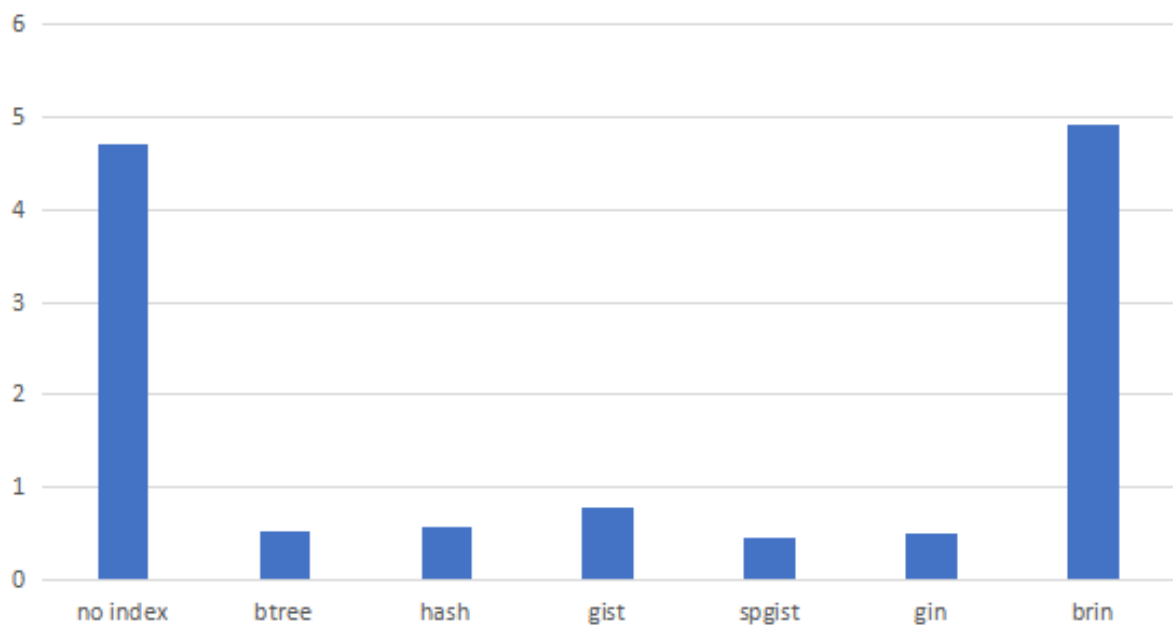


Рисунок 6 – Среднее время выполнения второго запроса

5. Анализ эффективности

Согласно QUERY PLAN, при выполнении первого запроса без использования индекса, сначала происходит Seq Scan или Table Scan, оператор получает строки из таблицы последовательно. Затем фильтр принимает входные данные и возвращает только те строки, которые удовлетворяют критерию фильтрации (`player1_name = 'Robert Horry'`). В результате выполнение запроса занимает 168,219 ms.

При использовании индекса В-дерево в запросе сначала происходит Heap Scan, и по условию (`player1_name = 'Robert Horry'`) находится 690 heap blocks. Затем выполняется Index Scan и возвращаются только те строки, которые удовлетворяют условию. В результате выполнение запроса занимает 1,282 ms.

При использовании индекса "BRIN" в запросе сначала происходит Gather, как и в случае, когда индекса нет. Затем выполняется Heap Scan, и по условию (`player1_name = 'Robert Horry'`) находится 12529 heap blocks, что значительно больше, чем при использовании других индексов. Затем выполняется Index Scan и возвращаются только те строки, которые удовлетворяют условию [8]. В результате выполнение запроса занимает 182,610 ms.

Первый запрос имеет одинаковый QUERY PLAN при использовании индексов В-дерево, Хеш, GiST, P-GIST и GIN, но немного различается время выполнения из-за небольших отличий в actual time of Index Scan [4-7]. GiST - это сокращение от "Generalized Search Tree" и представляет собой сбалансированное дерево поиска. В случае, если поддерживаются операторы "больше", "меньше" и "равно", GiST не так эффективен, как индекс В-дерево, поэтому результат индексации GiST в данном эксперименте хуже, чем у индекса В-дерево. Однако метод индексации GiST полезен для типов данных, которые не имеют смысла для этих операторов, например, географических данных, текстовых документов, изображений и т.д.

Заключение

В данной работе исследовались различные индексы для базы данных баскетболистов Национальной баскетбольной ассоциации (НБА), включая В-дерево, Хеш, GiST, SP-GiST, GIN и BRIN. Результаты исследования показали, что после создания индексов В-дерево, Хеш, GiST, SP-GiST и GIN эффективность поиска данных значительно увеличилась, а BRIN-индекс не подходит для данной базы данных. В работе была произведена анализ эффективности обработки запросов для разных типов индексов, основанный на QUERY PLAN [9]. Эксперименты показали, что В-дерево может работать с условиями на равенство и проверками диапазонов для данных, которые можно отсортировать в определенном порядке, а хеш-индексы работают только с простыми условиями равенства.

В будущем мы можем создавать более современные базы данных, которые будут хранить такие типы данных, как географические данные, текстовые документы, изображения и другие. Основываясь на полученных результатах, мы можем разработать новую систему индексации, которая будет более подходящей для нашей базы данных.

Список литературы

1. Фролов, К. М., & Казакова, Р. И. (2014). В-дерево как индексная структура данных. In *Информационные технологии в науке и образовании. Проблемы и перспективы* (р. 159).
2. Веретенников, А. Б. Эффективное создание текстовых индексов. *Проблемы теоретической и прикладной математики: Труды*, 39, 348-350.
3. Хайков, Д. В., & Пипенко, Р. С. (2022). СРАВНЕНИЕ СТРУКТУР ХРАНЕНИЯ ДАННЫХ «ДВОИЧНОЕ ДЕРЕВО», «КРАСНО-ЧЕРНОЕ ДЕРЕВО» И «В-ДЕРЕВО». In *НОВОЕ СЛОВО В НАУКЕ И ОБРАЗОВАНИИ* (pp. 7-10).
4. Иванова, Е. В. (2014). Использование распределенных колоночных хеш-индексов для обработки запросов к сверхбольшим базам данных. In *Научный сервис в сети Интернет: многообразие суперкомпьютерных миров. Труды Международной суперкомпьютерной конференции. М.: Изд-во МГУ* (pp. 102-104).
5. РЕДЮК, С., & САТТАРОВ, А. (2015). Реализация метода последовательного обхода GiST индекса в СУБД PostgreSQL. In *XIX Международная телекоммуникационная конференция молодых ученых и студентов "МОЛОДЕЖЬ И НАУКА"* (pp. 34-34).
6. Aref, W. G., & Iyas, I. F. (2001). Sp-gist: An extensible database index for supporting space partitioning trees. *Journal of Intelligent Information Systems*, 17, 215-240.
7. Баракнин, В. Б., Кожемякина, О. Ю., & Борзилова, Ю. С. (2020). Оптимизация SQL-запросов на примере работы поискового модуля системы комплексного анализа художественных текстов. *Cloud of Science*, 7(4), 749-763.
8. Меджидов, Р. Г. (2019). Анализ многоколоночных индексов баз данных. In *Актуальные проблемы прикладной математики, информатики и механики* (pp. 420-422).
9. Rizzolo, F., & Vaisman, A. A. (2008). Temporal XML: modeling, indexing, and query processing. *The VLDB Journal*, 17, 1179-1212.
10. Li, K., & Li, G. (2018). Approximate query processing: What is new and where to go? A survey on approximate query processing. *Data Science and Engineering*, 3, 379-397.